DOCUMENT RESUME

ED 305 905 IR 013 762

AUTHOR Brandt, Richard C.
TITLE Building Databases for the Computer-Based Memorization System.
INSTITUTION Utah Univ., Salt Lake City. Dept. of Computer Science.
SPONS AGENCY Naval Personnel Research and Development Lab., Washington, D.C.
PUB DATE 20 Sep 88
CONTRACT N00244-83-C-1759
NOTE 26p.; For a related report, see IR 013 763.
PUB TYPE Guides - Non-Classroom Use (055) -- Reports - Descriptive (141)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS Authoring Aids (Programing); *Computer Games; Computer Graphics; *Computer System Design; Courseware; *Databases; Higher Education; Models; *Networks; *Semantics; Specifications
IDENTIFIERS *Associative Networks; *Computer Based Memorization System

ABSTRACT

The Computer-Based Memorization System (CBMS), which specifies the facts that students are to know and how well the facts are to be known, uses a compiled form of an associative network for its knowledge database. (An associative network is a knowledge representation that uses associations for its basic representation of knowledge.) The CBMS provides eight computer games which use randomness and scoring to make the memorization more interesting. In addition, the games create and modify a file containing a model of the student's current knowledge of the material in the database. When the data in this file shows that a student knows a particular fact, the games stop asking questions about that fact. Three additions to the use of associative nets have been made in this system: (1) importance values had to be associated with items and statements to permit authors to emphasize features and support files containing models of students' knowledge; (2) pictures had to be associated with items; and (3) groups of anticipated answers had to be associated with items to allow a range of acceptable student answers. With these changes, computer-based memorization systems appear to be an effective tool for the mastery of factual knowledge. (4 figures and 12 references) (EW)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☒ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED305905

Building Databases for the Computer-Based* Memorization System

Richard C. Brandt
Computer Science Department
University of Utah
Salt Lake City, Utah, 84112

September 20, 1988

1 Introduction

Building databases for the Computer-Based Memorization System (CBMS) is different from creating ordinary script-based Computer-Aided Instruction (CAI). In script-based CAI, authors specify precisely the materials to be presented and the type of presentation. In CBMS, authors specify the facts that students are to know and how well these facts are to be known. In the domain of fact acquisition the creation of instructional materials using CBMS knowledge databases appears to be significantly less time-consuming than the creation of script-based CAI for the same purpose. In addition, the use of CBMS for memorization appears to be on a sounder theoretical basis than using ordinary script-based CAI.

CBMS uses an associative network for its knowledge database. An associative network is a knowledge representation that uses associations for its basic representation of knowledge. Objects in this representation are described by associating attributes with them. This paper describes the issues involved in creating and editing the file that defines an associative network. The CBMS games use a compiled form of this associative network.

1.1 CBMS Games

Students use eight CBMS games[1] to learn facts. The eight games use randomness and scoring to make the memorization more interesting. In addition, the games create and modify a file containing a model of the student's current knowledge of the material in the database. When the data in this file shows that a student knows a particular fact, the games stop asking questions about that fact.

*The development of the CBMS programs was supported in part by the Navy Personnel Research and Development Center Contract N00244-83-C-1759. ©1987 Richard C. Brandt NOTICE: This material may be reproduced by or for the U.S. Government for its internal uses.

[1]The Computer-Based Memorization System Student Manual[4] describes the games in detail.

IR013762

BEST COPY AVAILABLE

The games distinguish between recall and recognition answers. Students usually may enter (recall) the answer to a question. If they are unable to recall the answer, they can try to select (recognize) the correct answer from a list of possible answers.

Although the main purpose of the CBMS games is the acquisition of factual knowledge, different games have different purposes that help students develop a variety of procedural skills. The particular purposes of the eight CBMS games are:

Concentration. Concentration's purpose is to reinforce the *perception of relationships* among items. The game uses a matrix board on which each row and column corresponds to items in a selected category.

Constraint. Constraint's purpose is to help students learn to *discriminate among members* of a selected category. Students decide whether a given statement is true or false for each item in a list of category members.

Flashcard. Flashcard's purpose is to help students *recall all facts* about members of a selected category. Flashcard is an electronic version of conventional study aids based on flash cards.

Identify. Identify's purposes are to help students (1) *identify members* of a selected category from short descriptions and (2) *review facts* about the identified members.

Jeopardy. Jeopardy uses a matrix game board to teach students *to identify members* of selected categories from their complete descriptions.

Matrix. Matrix's purpose is to help students learn those items in the database *have similar properties.*

Picture Quiz. Picture Quiz's purpose is to provide students with practice in *recognizing objects from pictures.*

Twenty Questions. Twenty Questions' purpose is to give students practice in *selecting and creating discriminating questions.* Students ask questions with yes/no answers to eliminate all but a single item from a list of database items.

1.2 Other CBMS Programs

In addition to the games, there are three other CBMS programs.

Browser. Authors and students use Browser to examine CBMS databases. Browser shows the structure of a database and lists attributes of items in the database.

Fromtext. Authors use Fromtext to convert text representations of associative networks into forms that can be used by the CBMS games and Browser.

Totext. Authors use Totext to convert CBMS databases from the forms used by the CBMS games to a text representation that can be examined and modified.

BEST COPY AVAILABLE

1.3 Cognitive Science Basis

The associative net model of memory influenced both the design of the Tactical Gaming System (CBMS's predecessor) developed by Drs. Hollan, Crawford, and McCandless of the Navy Personnel and Research and Development Center[10, 6] and the design of the CBMS games. The following sections describe the associative net model and related recall experiments.

1.3.1 Associative Net Memory Model

The associative net model of memory provides the theoretical basis for CBMS.[2] Variants of this memory model provide useful descriptions of memory functions such as retention and retrieval.

The associative net model of memory[3] asserts that memory is constructed from elements (words, strings, pictures) interconnected to form cognitive units called traces. For example, assume the following fact is in long-term memory.

```
"Ronald Reagan" "was elected" president;
```

The *elements* are the three phrases

```
"Ronald Reagan"
"was elected"
president
```

and the *trace* connects the three phrases to form a fact. An element can be in many traces. For example, most people can recall many traces (facts) involving each of three elements above.

In the associative net model, a student will recall a trace or element in long-term memory only if there is a path connecting it through traces to elements in working memory. For example, if a student is trying to name the last five presidents, and the trace "Ronald Reagan was elected president" is in the student's long term memory, there is a certain probability that the student will recall the element "Ronald Reagan" because the element "president" is connected to the element "Ronald Reagan." In other words, since the trace "Ronald Reagan was elected president" is in long term memory of the student, the student may *associate* the element "Ronald Reagan" with the element "president."

In the associative net model, if an element or trace is in working memory and not yet in long-term memory, there is a certain probability (this probability depends on the individual) that a copy of the element or trace will be created in long-term memory. Furthermore, if an element or trace is in both working memory and long-term memory, the *strength* of the element or trace in long-term memory is increased. The recall probability of

[2] Associative nets have been used for describing memory since the initial work of Collins and Quillian.[5] They have also been used for substantially more complex systems. For example, Rumelhart and Norman[12] treated networks including nodes for events, concepts, and episodes. *Associative Networks* edited by Findler[8] contains a collection of essays on associative networks.

[3] Much of the information discussed in this section comes from John R. Anderson's book, *Architecture of Cognition*.[1]. The model discussed here closely follows the model used in Anderson's ACT* theory.

an element or trace is proportional to its strength. Additional separate occurrences in working memory of the element or trace further increase its strength. The strength of unused elements and traces slowly decreases with time at a rate that is dependent on the individual.

The recall probability of a trace or element is related to its *activation*. Activation is a measure of the "activeness" of a trace or element. If the activation of an item is large, its recall probability will be higher. The initial sources of activation are items and images in working memory that have been received from the external environment. The activation spreads from an active mode to its nearest neighbors (closest associates). For example, if the element "president" is in working memory, activation will spread to the elements "Reagan," "Carter," "Ford," "Nixon," "Johnson," and "Kennedy." The relative activation of these elements will be proportional to their strength. For example, if each of the elements has equal strength, the activation of each one will be equal; or if an element has much greater strength than the other elements, its activation will be much greater than that of the other elements. The amount of activation that an element can provide is limited and independent of the number of its closest associates. For example, call the amount of activation "A." If an element has just one associate, it will receive the entire activation "A." If an element has five associates, each of equal strength, each will receive a fifth of the activation, "A/5." This is called the *fan* effect.[1]

Activation spreads through an associative network from neighbor to neighbor. An element or trace may receive activation from several neighbors. For example, assume a student is trying to recall the name of the man "who was elected president in 1800 and wrote the Declaration of Independence." Then the elements in working memory that serve as sources of activation are:

```
president
1800
"Declaration of Independence"
```

These all are associates (nearest neighbors) of the element "Jefferson" and each will contribute to the activation of the element "Jefferson."

1.3.2 Recall and Recognition

Recall and recognition experiments have demonstrated that the length of time that a trace is continuously in working memory has little effect on its later recognition or recall.[11] Consequently, there is no point in having students repetitively answer the same question in the CBMS games. The CBMS games use a two-in-a-row game rule to avoid this. Specifically, if a student correctly answers a particular question two times in a row[4],[5] the game will not ask the question again.

[4]Nelson[11] demonstrated a small but not insignificant positive effect of repetition within a test session, particularly if there were intervening questions.

[5]There may be, and almost always are, intervening questions in a game.

Recall answers are generally more difficult for students for two reasons. First, in the associative net model, there is less activation for a recall answer than a recognition answer because a students see one more component (the answer) of a fact in a recognition answer. Second, recall answers in games require that students type the answers. Because of these added difficulties, the CBMS games give more points for recall answers.[6]

Past experiments have shown that practice leads to greater accuracy and shorter recall times if there are many practice sessions. For example, Anderson[1] (p. 183) found that the recognition time for sentences consisted of two components: an intercept time independent of the number of practice sessions and a component inversely proportional to the number of practice sessions to the 0.36 power; that is

$$T = Intercept + Constant \star P^{-0.36}$$

where P is the number of practice sessions. In CBMS, the use of the two-in-a-row rule and an "Importance" file permits authors to control the minimum number of separate practice sessions in which students will have to answer questions involving a database fact or item correctly. A number called "Importance" is associated with each item and fact in a CBMS database. Each time a student answers a question involving a particular item or fact correctly, one is subtracted from its Importance; each time a student answers a question incorrectly, one is added to its Importance. The probability that a particular item or fact will appear in a question is proportional to its Importance. When the Importance of an item or fact reaches zero, it will not appear in questions. The two-in-a-row rule keeps the Importance of an item or fact from going down more than two during any game session. This allows an author to specify in advance the minimum number of practice sessions that will include a particular item or fact. For example, if an author initializes the Importance of an item or fact at 10, the students will see questions involving the item or fact in at least five ($5 = 10/2$) separate sessions.

[6] During a game students may obtain more points by selecting recognition answers simply because it takes less time to move the cursor to a correct answer in a menu than to type a recall answer.

1.4 CBMS Databases

CBMS databases are associative networks that contain the material to be memorized. As such, they contain statements and items. Statements correspond to traces in memory and items correspond to elements in memory.

1.4.1 Statements

CBMS databases contain two types of statements:

- Categorization statements. A categorization statement specifies the category to which an element belongs. An element may belong to only one category.
- Attribute statements. An attribute statement specifies a property of an

element in the databases.

Categorization probably plays a more important role in CBMS than it does in memory. The category membership statement in CBMS is "isa." Categories are important in CBMS games because to play, students select one or more categories for study. Categorization in CBMS is stricter than it is in memory. Each item in CBMS must belong to one and only one category. For example, CBMS prohibits having the following two statements in the same database.

```
"Ronald Reagan" isa Republican;
"Ronald Reagan" isa president;
```

The category structure in CBMS is similar to biological classification including inheritance of attributes. For example, assume the following three classification statements are in the database.

```
hound isa dog;
beagle isa hound;
Snoopy isa beagle;
```

In CBMS "Snoopy" will inherit the attributes of dogs, hounds, and beagles. For example, "Snoopy"will inherit the attributes given in the following statements:

```
dog has "4 legs";
dog eats meat;
dog "related to" wolf;
hound ears drooping;
hound voice deep;
hound "follows prey by" scent;
beagle coat smooth;
beagle legs short;
beagle size small;
```

Because of inheritance[7], categories may only have attributes that characterize all or a least most of their members. Because of inheritance, authors need to associate directly with a given item only those attributes that distinguish it from other members of the same category.

1.4.2 Items

CBMS databases contain four types of items:

- Words. A word is a single word.
- Strings. A string consists of an *ordered* list of words. For example, the phrase "Ronald Reagan" is a string.
- Pictures. A picture may contain graphics, text, and video.
- Alternative Answers. An alternative answer is a list of words and formatting statements that define acceptable student answers associated with an item.

[7] In CBMS, authors can prevent the inheritance of particular attributes. For example, it is possible to state that "birds can fly" and then prohibit penguins from inheriting that attribute.

Alternative answers are used in the analysis of students' recall answers.

1.4.3 Constraints

A CBMS database is not as expressive as other associative networks because it contains only three-part statements. For example, it is not possible to express in CBMS the following:

```
Snoopy followed Molly "to the park";
```

The restriction to three-part statements simplifies question generation in the games. Question generation is simple using three-part statements,[8] but it is difficult with four-part statements.

There is an additional practical restriction. Because students must recall and enter answers that are one part of a three-part statement, these parts cannot be too complicated. CBMS works better with short answers than with long ones.

2 Specifying the Knowledge

Specification of the facts to be mastered using a CBMS database is similar to ordinary instructional design.

[8]Authors obtain questions with a particular form by associating question templates with the "relations" in a CBMS database.

2.1 Objectives

The first step in creating a CBMS database is to define the objectives and the intended audiences. What are students expected to know? This definition process is far more specific and restrictive for CBMS than for ordinary instruction. For example, the objective of a beginning physics course might be mastery of elementary mechanics. In the instructor's mind this includes both procedural and factual material. This type of specification is too general for CBMS. A more acceptable specification would be "the names and attributes of around 40 basic formulas in elementary mechanics and the names and attributes of around 10 problem-solving procedures.[9]" A specification of an expected audience could be "university freshmen and sophomores who have already taken one course in calculus." Another example of a specification acceptable for CBMS would be "200 facts on cranial nerves for first-year medical students in a neuroanatomy course."

Initial objectives may not be too specific because instructors often don't know how many facts they are currently presenting to students. However, setting an initial limit on the number of facts helps avoid the problem of too large databases.

[9]CBMS requires names for all items and that everything be explicit. For example, this requires a name for a problem-solving procedure, the specification of the criteria for using it, and the listing of its attributes. This is unlike the standard "example-driven" teaching in physics where an instructor will show a problem solution, but will not name the problem-solving procedure or explicitly identify the criteria for its use. Often students must "discover" these criteria by doing homework problems.

2.2 Data Collection

Facts may be obtained from various sources. Example sources are:

- textbooks
- reference books and user manuals
- teachers and instructors
- experts

In the process of collecting facts, the objectives will become more specific. Usually, the items that will be subjects of questions will become well defined. For each item, there will be a list of attributes that characterize it.[10] The initial lists of attributes probably will include facts that may not seem absolutely necessary for the student to know. Initial inclusion of these facts does not hurt, however, they should be marked for possible later exclusion. The initial lists should not include facts that students clearly do not need to know.

A "naming" problem may occur at this stage if there are several names for an item or there is no name for an item. If an item has several names that should be known by students, the names may be combined. For example, the first cranial nerve is called both "Olfactory Nerve" and "Cranial Nerve I." The two names may be combined to form "Olfactory N./CN I." A more difficult problem is the absence of a name for an item. For example, if the item is a particular problem-solving procedure in physics, the procedure may have no name. Two solutions are possible; either a place-holding name such as "ProcedureC" may be used, deferring the problem, or a name may be made up. Since CBMS permits phrases as names, made-up names can be descriptive.

[10] Items cannot be distinguished by name alone in CBMS.

2.3 Common Features

Features common to many items will usually appear in the initial fact collection. For example, most cranial nerves:

- have at least one function
- enter the cranium through a foramen
- have at least one functional component
- have cell bodies in a particular location

Common features of a database item are easier to spot if its attributes are listed immediately after it in alphabetical order. Errors of omission can be detected by reviewing the common features. For example, a review of a database may show that many but not all items have certain features. Should all items, or at least all similar items, have these features? If they should, then more data must be collected.

2.4 Categorization

The categorization of items when the subject matter has an already defined structure is easy; the categorization can follow the existing structure. For example, most biological systems are already classified. When there is no usable predefined structure, the items may be categorized using the following rules.

1. Items with common features belong in the same category.
2. A category usually should have only features common to its members.
3. The categorization should be shallow; that is, a student selecting a category to study should not have to make more than one or two choices.

2.5 Expert Review

After the subject matter has been organized into a database, other subject matter experts should review it. Generally, they will address the following questions:

- Are there items omitted that should be included and vice versa?
- Are there facts omitted that should be included and vice versa?
- Is the categorization reasonable or are there better categories?

Some disagreement is to be expected. In particular, there is often disagreement between individuals who use the information in the field and those who teach it.

2.6 Revision

Before the database is revised, a limit should be set on the number of facts that students will be expected to know. Authors sometimes make the mistake of including in the database all facts that the experts think are important. This ignores the fan effect, which implies that the more attributes an item has, the more difficult it will be to recall any particular one.

It is important to identify and include those facts that experts use in item recognition and in decision-making. A recent study by Grant and Marsden on medical diagnosis[9] revealed that differences in diagnostic expertise between specialists and medical students could be explained by identifying facts that the specialists felt were most important ("forceful facts") in the diagnosis.

3 Implementation

Implementing a CBMS database includes four steps:[11]

- creating and editing the defining text file with an ordinary text editor
- creating and editing pictures with the sequence editor
- compiling the files with Fromtext
- evaluating the databases with Browser and the game programs

3.1 CBMS Text Files

CBMS text files may be edited and created with any text editor that produces an ASCII text file.

[11] Implementation of CBMS databases is described in detail in the Computer-Based Memorization System Author and Instructor Manual[3].

3.1.1 Form

The categorization and attribute statements in a CBMS text file consist of three parts:

1. subject
2. relation
3. object

Each part may be a word or a string. A string is just an ordered collection of words or numbers; for example, the phrases "Ronald Reagan" and "4 wheels" are strings. Double quotation marks mark the beginning and end of strings. Statements always end with a semicolon. Example statements are:

```
"Ronald Reagan" "was elected" president;
"abducens nucleus" isa nucleus;
```

Occasionally the second part of a statement, the relation, corresponds to a verb.

Categorization All items in CBMS databases belong to a category. An item belongs *directly* to a category if there is a categorization statement of the form:

```
item_name isa category_name;
```

An item is an *indirect* member of a particular category if it is a member of another category that is either a direct or indirect member of the particular category. There are four predefined categories: the most general category to which all items in the database indirectly belong, *db_item*, and the three categories:

```
db_element
db_relation
db_object
```

All items that are to be subjects of questions must belong either directly or indirectly to the category *db_element*. A member of the category *db_element* may be either the subject or object in statements. Items that may be either the subject or the object in statements but *not* the subjects of questions belong to the category *db_other*. Relations must belong to the category *db_relation*.

Attributes Attributes describe items in the database. The complete description of an item consists of attributes specific to the item and attributes that characterize the categories to which the item belongs. For example, the attributes specific to a beagle are:

```
beagle coat smooth;
beagle legs short;
beagle size small;
```

Beagles also have the attributes of hounds and dogs since they are members of the categories hounds and dogs. Note that the preceding statements are terse and the relations in the statements restrict the possible objects. In CBMS, the question formats that specify how questions are to be generated, are associated with the relations. Questions that might be generated from the preceding three statements are:

```
What type of coat does a beagle have?
```

```
What length (short, medium, long) legs
  does a beagle have?
How large a dog is a beagle?
```

If a more general relation such as "has" is used, the questions cannot be as specific. For example, assume the attributes specific to a beagle are:

```
beagle has "smooth coat";
beagle has "short legs";
beagle has "small size";
```

Then only one general question is possible.

```
What does a beagle have?
```

This question with its three answers is more difficult than the preceding three specific questions.

The games also use relations in hint generation. The alternatives presented in the hint list are other objects of a relation. For example, other objects of the specific relation "coat" might be: "heavy," "rough," "short," "wire-haired," and "long." The use of closely related alternatives is often desirable. The generation of these alternatives depends on the use of the same relation. For example, if an author uses the relation "hair type" and the relation "coat", the objects of "hair type" will not be used as hints for an object of a relation using "coat."

A further advantage to the use of the same relation when possible (for example, rather than using "coat" and "hair type" using only "coat") is that its use helps create alternative paths to an object in the mind; these help students remember objects. For example, if students cannot remember the type of coat a beagle has they might think that a beagle is like a fox hound and has a similar coat.

If there is no problem with ambiguity, relations can be terse. For example, "coat" is a terse relation. Terse relations require less typing and often are easier to locate in a database. However, terse relations may result in a database that is hard to read and to evaluate.

3.1.2 Organization

It is easier to review a database that has a consistent structure. The alphabetical outline method is one reasonable way to organize an database. In this method, all items belonging to *db_element* are placed first, all items belonging to *db_other* are placed second, and all items belonging to *db_relation* are placed last. For each major category, the items and categories that are its direct members are placed in alphabetical order. Finally, the attributes of each item are listed in alphabetical order. Indentation similar to that used in ordinary outlines may be used.

If there is some natural order, then this should be used when appropriate. For example, the twelve cranial nerves are listed in numerical order.

Categories in the database should not have too many direct members. There are two problems with having too many direct members; first,

the fan effect will reduce the activation of each member and second, impossible and unfair questions may be generated. For example, one experimental database resulted in the instruction:

```
Name the Democrats in Congress.
```

3.1.3 Sentence Templates

Sentence templates are patterns used to convert attribute and classification statements into forms that can be used in the games. They are necessary because questions directly constructed from attribute or classification statements may sound awkward or be difficult to understand. Sentence templates are associated with the relations in a database. There are four types of sentence templates:

- Statement
- Subject
- Object
- Yes_No

Statement templates are used to convert database statements to acceptable English. For example, a Statement template may convert the database statement, "beagle coat smooth," to "The coat of a beagle is smooth."

Subject templates are used to create questions when the answer will be the subject of a database statement. For example, a Subject template may generate the question "What type of hound has a smooth coat?" from the database statement, "beagle coat smooth."

Object templates are used to create questions when the answer will be the object of a database statement. For example, an Object template may generate the question "What type of coat does a beagle have?" from the database statement, "beagle coat smooth."

Yes_no templates are used to create questions with yes or no answers from a database statement. For example, a Yes_no template may generate the question "Does a beagle have a smooth coat?" from the database statement, "beagle coat smooth."

Because sentence templates are associated with relations, they cannot explicitly mention either the subject or object of a particular database statement. Instead special symbols represent the three general components of database statements.

"%s%" stands for the subject in a database statement, except when it is used in a Subject template. In a Subject template %s% stands for the category to which the statement subject belongs.

"%o%" stands for the object in a database statement, except when it is used in a Object template. In a Object template %o% stands for the category to which the statement object belongs.

"%r%" stands for the relation in a database statement. Since sentence templates are associated with relations, use of %r% is necessary only when sentence templates are shared by several relations. Templates associated with a category of relations are "inherited" and hence shared by the category

members. For example, if the category "dog feature" has the members "coat," "muzzle," and "tail," these members will share sentence templates associated with "dog feature."

Examples of sentence templates that may be associated with the relation "coat" are:

```
coat STATEMENT "The %r% of a %s% is
  %o%.";
coat SUBJECT "What has a %o% %r%?";
coat SUBJECT "What type of %s% has
  a %o% %r%?";
coat OBJECT "What type of %r% does
  a %s% have?";
coat YES_NO "Does a %s% have a %o%
  %r%?";
```

These sentence templates will generate the example sentences and questions on beagles contained in this section (p. 12) when "beagle" replaces %s% (in the second Subject template, "hound" replaces %s%), "coat" replaces %r% and "smooth" replaces %o%.

If a database is to be used with all the games and Browser, each of the four types of sentence templates must be associated directly or indirectly through inheritance with each relation in the database.

Creating Templates A sentence template for a question is formed by (1) creating a question from a database statement and (2) replacing the instances of the subject in the question with %s% and the instances of the object with %o%. Statement templates are created in a similar manner. Questions must be carefully written so that they are not ambiguous. Students regard ambiguous questions as unfair. Since the templates are associated with relations, the relations do not have to appear in templates. Often using words different from the words in the relation will result in a clearer question or statement. If questions or statements cannot be made clear for all potential objects or subjects, it may be necessary to introduce additional relations.

The articles used in the templates must be chosen with care. If none of the objects or subjects that appear after an article either start with a vowel or are plural, use of the article "a" is acceptable; otherwise, it is better to use the article "the."

3.1.4 Anticipated Answers

Anticipated answers are specified by using the MATCHED_WITH relation in statements. For example, the statement:

```
"Oculomoter N./CN III" MATCHED_WITH "CN
  III | oculomotor | cranial nerve III";
```

associates the anticipated answer "CN III | oculomotor | cranial nerve III" with the database item "Oculomotor N./CN III." Students' answers are first compared exactly against the item's name ("Oculomotor N./CN. III" in the preceding example). If a student answer matches exactly, it is correct. If a student answer does not match an item's name and an anticipated answer has been specified, the student answer is compared with the anticipated answer. If they match, the student answer is correct.

The use of anticipated answers makes the games much more acceptable to students and distinguishes CBMS databases from ordinary associative nets that do not use anticipated answers. Because the answer analyzer strips punctuation marks from student answers, anticipated answers do not contain punctuation marks. Anticipated answers and the ways in which they are matched against student answers are described using the following terms.

Tokens Anticipated answers are constructed from special symbols and tokens. A token may be a number or an identifier. *There must be a space on both sides of a token unless it is the first or last token in the anticipated answer.*

A **number** is a sequence of digits that may include a few special characters. Examples are: "122," "1000, " and "1.34e-15." The default tolerance is ±3%. For example, if an anticipated answer is 100, student answers between 97 and 103 will match. A different tolerance is specified by including after a number a percentage. For example, an anticipated answer of 100 %1 will match student answers between 99 and 101, inclusive. A range may be specified instead. A range is specified by a lower bound followed by two dots (periods) and then an upper bound. For example, if the anticipated answer is 101 .. 103, student answers between 101 and 103 inclusive will match.

An **identifier** is any sequence of letters and digits that is not a number. Examples of identifiers are: "ball," "operation," and "1ff10d." Identifiers in anticipated answers may include a spelling tolerance. The default tolerance is 20%. A different tolerance may be specified by placing a percent symbol after the answer followed by the tolerance percentage. A 0% tolerance requires exact matches. A spelling checker computes a score when it compares two words and compares this score with the tolerance. The spelling checker assigns different weights to different errors. For example, the initial letter of each word counts twice as much as the other letters and a transposition of characters counts as one third of a letter that disagrees. The spelling checker will remove the last "s" from a student answer if doing so improves the match. Examples of anticipated identifiers are: "trip," "helicopter %0," and "gadhafi %50."

Modifiers Modifiers are special symbols used as prefixes for tokens in anticipated answers so that student answers will be interpreted in special ways. There are four modifiers.

@c means that the cases of the letters (upper-case and lower-case) in a student's answer are important. Normally the cases are ignored. For example, the answers trip, Trip, and TRIP match the anticipated identifier trip. If the anticipated identifier is @cTrip %0, then only the answer Trip will match it. A letter in the wrong case counts as a letter that disagrees.

@l means that the number following @l is to be interpreted as a character sequence rather than as a number. For example, only the

answer 5 will match @15; the answer 5.00 will not. This format is useful in counting and with numbers used for identification rather than as numbers; for example, Boeing @1737.

@p means that the token is to be used as a pattern rather than as a separate token. A word will match the token if a portion of the word has the same pattern. For example, the pattern @pamp will match the student answers amps, amperes, amp, damp, kiloamps since in each case the pattern amp is present. Patterns are case senstive; for example, @pamp will not match Amp.

@ means that the identifier following @ is to be interpreted as a character sequence and that no further processing to be done on it. This permits anticipated answers to contain special symbols that are used in answer specification. For example, the pattern @(expression) will match the student answer (expression).

There is no space between a modifier and the token it modifies.

Words A **word** is a token or a token prefixed with a modifier. Examples of words are: "helicopter" and "@cWashington."

Wordlists and Match Answers A word list can be a single word or a list of words separated by spaces. Examples of word lists are:

```
five @15
a the
helicopter
```

Word lists are used in "Or" anticipated answer parts and "Optional" match anticipated answer parts.

An Or answer part consists of the special symbol **@o** and a word list inside parentheses. If a student's answer contains one item in the word list it will match the Or part.

The Optional answer part consists of the special symbol **@?** and a word list inside parentheses. An example is "@?(a the)." If a student's answer contains one or none of the words in the word list it will match the Optional part.

An example of an anticipated answer containing an Or and an Optional part is: "I like @?(a the) @o(red white) rose". It matches the following student answers:

```
I like red roses.
I like the red rose.
I like a white rose.
```

Since the spelling checker removes the last "s" from tokens if that will improve the match, the first student answer matches although it contains "roses" instead of "rose."

Range and Anything Match Answers Anything match answers, **@a** answers, are appropriate when the particular tokens in the student answer are not important; only the number of tokens is important. Three methods for specifying ranges are supported with "@a": an asterisk, *, specifies a range of zero or more tokens; a plus sign, +, specifies a range of one

or more tokens; and two numbers separated by a comma, **m, n**, specify a range explicitly. For example, the anticipated answer "a big @a* ball" will match the following student answers:

```
a big ball
a big red ball
a big red and white ball
```

The tokens "red", "and", "white", match @a*. An example of the use of an explicit range is "a big @a1,3 ball". It matches the first two but not the second two of the following four answers:

```
a big red ball
a big red and white ball
a big ball
a big red white and blue ball
```

Part A part of an anticipated answer may be a word, an Anything match of a range of words, an Or match of one word in a word list, or an optional match to one word in a word list. For example, in the anticipated answer "I like @?(a the) @o(red white) rose" the parts are:

```
I
like
@?( a the )
@o( red white )
rose
```

Partlist A partlist is a list of one or more parts. An example of a partlist is:

```
I like @?( a the ) @o( red white ) rose
```

A partlist is matched if each of its parts match corresponding parts in the same order in the student answer.

Pattern A pattern is a partlist that may be marked at the beginning with **@b** or at the end with **@e**. If @b marks the beginning of a partlist, the text after @b must be at the beginning of a student answer. If @e marks the end of a partlist, the preceding text must be at the end of a student answer.

For example, the anticipated answer

```
@b I like @?( a the ) red rose
```

will match the student answers:

```
I like red roses and purple violets.
I like the red rose that won first
  place in the contest.
```

but not the answers:

```
In the park, I like the red roses best.
I like violets and I like red roses.
```

If neither @e nor @b is present, the *partlist may appear anywhere in the student's answer.* If both @e and @b are present, a student answer containing the anticipated answer part alone is the only matching answer.

Excludes Occasionally, it is useful to look for the absence of some pattern in a student answer. Preceding a pattern with the minus sign, -, excludes it. For example, the following anticipated answer

```
@?( a the ) red rose -@o( not @pn't )
```

will match the following student answers:

```
I like red roses
he does like red roses
```

but not:

```
I do not like red roses
he doesn't like red roses
```

The excluded wordlist "not @pn't" provides a way of handling single negatives in student answers.

Patternlist A patternlist is a list of patterns separated with the special symbol "|" and optionally followed by a group of patterns that are excluded. For example, the anticipated answer

```
@?( a the ) red rose -@o( not @pn't )
```

is a patternlist because it contains an excluded pattern. The first pattern is

```
@?( a the ) red rose
```

and the second pattern (which is excluded) is

```
@o( not @pn't )
```

Technically, all anticipated answers are patternlists; usually, they will consist of only one pattern. An example of a patternlist using the special symbol "|" is

```
He likes red roses | She likes violets
```

Note that there must be a space on both sides of the special symbol "|". "|" separates the patterns in a patternlist. Students' answers match if they match a pattern in the patternlist.

3.1.5 Changing Default Values

Default values are used for items such as the tolerance to simplify editing. The default values can be changed or overridden by including appropriate statements in a CBMS datafile.

Default Tolerances The default tolerances for numbers (3%) and strings (20%) may be changed. The change will affect following items in the file. For example, the statement

```
DEFAULT_NUMERICAL_TOLERANCE 10;
```

changes the default numerical tolerance from its current value to 10%. The statement

```
DEFAULT_STRING_TOLERANCE 15;
```

changes the default string tolerance from its current value to 15%. Default tolerances may be changed at several places in a CBMS datafile.

Specifying Importances The Importance of an item or a fact determines the *net* number of times a student must correctly answer questions involving it before it is "known." The games do not ask questions about "known" items and facts. The net number is the number of correct answers minus the number of incorrect answers. For example, if the fact:

```
beagle coat smooth IMPORTANCE 10;
```

is included in a CBMS database text file, the games will continue to ask questions involving it until the number of correct answers to questions involving it entered by a student exceeds the number of incorrect answers by 10.

The Default Importance is used when the Importance of an item or fact is not explicitly specified as above. Initially it is 3. Its value is changed by including a statement specifying a new value for it. For example, the following statement changes it to 10:

```
DEFAULT_IMPORTANCE 10;
```

The Importance of the items and facts following this statement in the database will be 10 unless specified otherwise.

Inheritance The games use attribute inheritance, that is members of a category by default "inherit" the attributes of the category. For example, if beagles belong to the category of hound, beagles will "inherit" the attributes of hounds (drooping ears, deep voice, etc.). Sometimes, it is desirable for a category to have attributes that are not inherited by all the category members. This is done by placing the word "NO_INHERITANCE" after a statement of the attribute that would normally be inherited by a category member. For example, the statements

```
bird ability fly;
penguin isa bird;
penguin ability fly NO_INHERITANCE;
```

assert that all members of the category birds except penguins have the ability to fly.

Synonyms SYNONYM_FOR is a predefined relation that associates either a word with its synonym or a unit with a number and a related unit. Examples are:

```
in SYNONYM_FOR "1 inch";
in SYNONYM_FOR "2.54 cm";
feet SYNONYM_FOR "12 inch";
foot SYNONYM_FOR feet;
yard SYNONYM_FOR "3 feet";
meter SYNONYM_FOR "100 cm";
```

When a student plays a game, the game uses the conversion tables specified with the SYNONYM_FOR statements. These convert units appearing in student answers. For

example, if the anticipated answer is "1000 yards" and a student enters "3000 feet," the student's answer will match the anticipated answer if the preceding SYNONYM_FOR statements are in the database.

Hidden By default the games will present all possible questions. Occasionally it is undesirable to generate questions involving a particular item or relation. Often it is desirable to hide the "isa" relation because the categorization information can result in trivial questions. For example, in a database on cranial nerves, the class membership question "What is the abducens nerve?" appears with the trivial answer of "cranial nerve" because the database contains the categorization statement:

```
"Abducens N./CN VI" isa "cranial nerve";
```

Class membership questions will not appear if the following statement is in the database:

```
isa HIDDEN;
```

Including this statement "hides" the relation "isa" and blocks the presentation of any class membership questions. In general, placing the word "HIDDEN" after a relation or an item, "hides" it. The games do not generate questions involving a hidden relation or item.

3.2 Pictures

Pictures including graphics, video, and text may be associated with items in the database. Pictures, which consist of a linear sequence of text, graphics, and video frames, are called *sequences*. In CBMS datafiles, a picture is identified by naming the file containing the picture and then associating the picture with an item. A statement consisting of the word "USE_DISPLAYFILE" followed by the name of the file containing the pictures identifies the file. Since pictures are always stored in sequence files, the file name extension, ".SEQ," is omitted. For example, the following statement identifies the file "nerves.SEQ."

```
USE_DISPLAYFILE nerves;
```

Pictures are associated with items by including a statement consisting of the word "USE_DISPLAY" and the name of the sequence that is the picture. For example, the following statement associates the sequence "abducens" with the item "Abducens N./CN VI."

```
"Abducens N./CN VI" USE_DISPLAY
   abducens;
```

Authors use the Sequence Editor,[12] **se**, to create and edit sequences. The Sequence Editor is started by entering in response to the system prompt "se" followed by the name of the file to be edited or created. For example, to edit or create the file nerves,

```
se nerves
```

[12]The Sequence Editor is described in detail in the Sequence Editor Manual[2].

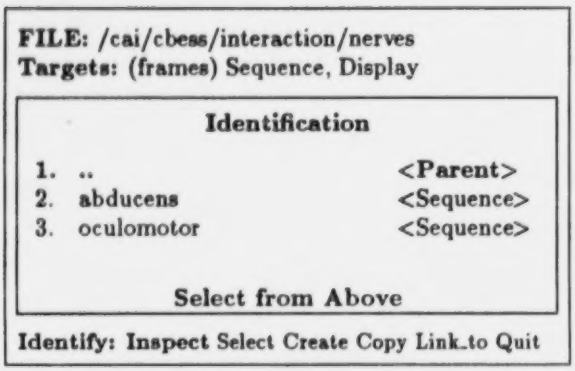


Figure 1: Identification Menu

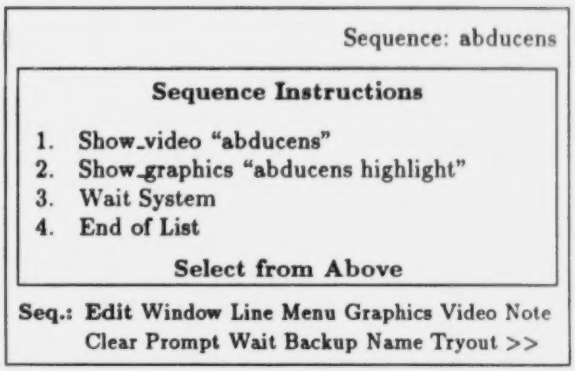

Sequence: abducens
Sequence Instructions
1. Show_video "abducens"
2. Show_graphics "abducens highlight"
3. Wait System
4. End of List
Select from Above
Seq.: Edit Window Line Menu Graphics Video Note
Clear Prompt Wait Backup Name Tryout >>

Figure 2: Sequence Instruction List

is entered in response to the system prompt. The screen will be cleared and the Identification Menu (see Fig. 1, p. 20) will be displayed.

The Identification menu lists the sequences in the file and the "parent" of the file that is represented by two dots. A sequence is created by choosing the command Create and then entering its name. A sequence is edited by moving the cursor to the sequence's name (use the up and down arrow keys or CTRL P and CTRL N) and then choosing the command Select. After a sequence is selected or created, the screen is cleared and the Sequence Instruction Menu (see Fig. 2) will be displayed.

The Sequence Instruction Menu lists the sequence instructions. A sequence instruction usually consists of a command that identifies the type of frame and the frame name. For example, the instruction

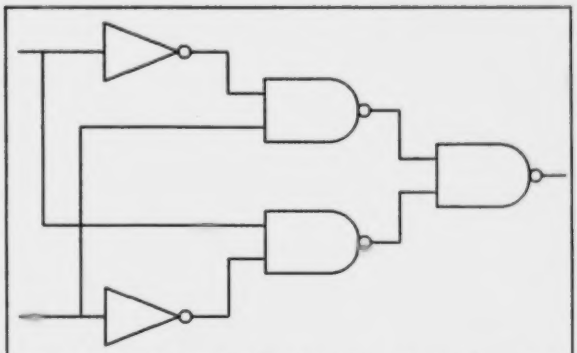
```
Show_graphics "abducens highlight"
```

specifies the display of the graphics frame named "adbucens highlight." The commandline at the bottom of the screen lists the commands. Only the following three commands are described here: Edit, Graphics, and Video. An instruction is edited by moving the cursor next to the instruction and then choosing the command Edit. A graphics or video instruction is created by choosing the command Graphics or Video and then entering the name for the graphics or video frame.

Graphics When a graphics frame is edited or created, the screen is cleared and the main graphics commandline along with the graphics is displayed (see in Fig. 3).

Authors use the graphics command Edit to create and edit graphics. When the Edit command is selected, the Edit commandline shown below is displayed. This commandline lists the graphics objects that may be used:

EDIT: Line Circle Arc Curve Fill Zone Symbol



FRAME: Edit Undo Working View Name Clear Quit
Status Modify Reorder Copy Examine >>

Figure 3: Graphics Editing

Group Delay Bound Quit

Symbols are user-defined graphics objects that are stored in a library. For example, in the graphics shown in Fig. 3, the triangular objects are symbols called "inverters" and the objects with the semicircular ends are symbols called "Nand2 gates."

Selecting a graphics object adds graphics to a picture. When an object is selected, new commandlines appropriate to that object are displayed. For example, when Line is selected, the following commandline is displayed:

LINE: Add Delete Relocate Move Rotate Extend Zoom Undo Color Background Index Pause Default Line Circle Arc Curve Fill Zone >>

Graphics objects are created by placing points on the screen using the Add command. For example, if Line has been selected and points are added, a sequence of lines will be drawn through the points.

Video: abducens

Video Control Frame

1.	**Start Frame**	**1298**
2.	End Frame	1298
3.	Play Speed	Normal
4.	Video Image	On
5.	Audio Channel	Both
6.	Final Video State	On
7.	Frame # Display	Off
8.	Student Control	None
9.	Videodisc Name	<none>

Select from Above

VIDEO: Edit Tryout Clear Undo Name Quit

Figure 4: Video Control Frame

Video When a video frame is edited or created, the screen is cleared and the Video menu and commandline (see Fig. 4) will be displayed.

The Video Control Frame contains the data necessary to specify the display of the video. An item is edited by moving the cursor next to the item and then choosing the command Edit. For example, to edit the Start Frame, move the cursor next to it and choose the command Edit. The Video commandlines will be displayed.

Halt_On Halt_Off Next Slow Normal Fast Browse Jump Direction Display Audio Keep Enter >>

The commands may be used to find the desired video material. If the frame number is already known, the Enter command may be selected to enter it.

3.3 Compilation

Authors use the program Fromtext to compile CBMS textfiles to forms usable by the CBMS games. Fromtext is started by entering the word "fromtext" followed by the name of the CBMS text file. For example, entering the following line in response to the operating system prompt

```
fromtext cranial.txt
```

will compile the CBMS textfile "cranial.txt." While Fromtext compiles a text file, it prints on the screen error messages and data that explain what it is doing. Three flags may be used with Fromtext: the flag 'q' turns off the screen display; the flag 'd' prints debugging information on the screen; and the flag 'o' specifies that any preceding file with the same name is to be overwritten. For example, to compile the CBMS textfile "cranial.txt" and to display debugging messages

```
fromtext -d cranial.txt
```

is entered in response to the operating system prompt.

Fromtext places the error messages in a file with the same name as the source file with the extension is ".ERR." For example, the file "cranial.ERR" will contain the error messages generated from the compilation of "cranial.txt." CBMS files should not be used with the games if any errors other than those associated with missing sequence files are detected.

The program Totext may be used to convert CBMS databases from the form used by the games to a text representation. Because Fromtext discards comments, no comments will be present in the text file generated by Totext.

3.4 Testing

Fromtext detects the syntactic errors in CBMS textfiles. These errors must be corrected before testing for errors is begun.[13] The remaining errors can be found by using the games and Browser. Browser may be used with the flag 'b' to detect placement problems and poorly worded statements. For example, to detect problems with the database "cranial"

```
browser -b cranial
```

is entered in response to the operating system prompt. The flag 'b' permits examination of the db_other and db_relation categories. Because Fromtext distinguishes between nodes by the spelling of their names, a common problem is the accidental misplacement of an item because of a spelling error. A version of the item with one spelling will appear where it

[13] Error messages resulting from missing sequence files can safely be ignored.

was anticipated, the other version will be placed in the db_other category "Not placed." Careful examination of the "Not placed" category will disclose these problems.

Other problems and errors are detected by playing the games. Common problems are not providing anticipated answers and misspelling item names.

4 Revision

Once a CBMS database is running and appears to be error free, it should be reviewed. This review supplements the initial reviews of the subject matter and addresses three issues:

- Matched_with statements
- Assigning importance
- Item exclusion

4.1 Matched_with Statements

The use of anticipated answers specified with MATCHED_WITH statements permits some latitude in student answers. However, the latitude is subject-matter specific; the range of an acceptable anticipated answer needs to be specified by experts in the field. Anticipated answers can be evaluated by having experts play the games and enter answers. Where their answers are not matched by an anticipated answer, the anticipated answer must be modified.[14] The problem of specifying an acceptable anticipated answer is similar to the problem of recognizing computer commands[7] with one major difference. While the objective in recognizing computer commands is to recognize the commands the user enters, the objective in recognizing anticipated answers is to accept only those answers that are acceptable to experts in the field. For example, "cranial nerve vii" is considered acceptable by neuroanatomists, but "cranial nerve 7" is not.

4.2 Assigning Importance

The Importance of items and statements in a database determines the minimum number of sessions in which students will have to answer questions involving the item or statement. One way to assign Importances is to divide the items and statements into three groups: critical, important, and needed but not important. The Importances can then be computed by determining the maximum number of separate sessions in which students can reasonably be expected to use the database; this number times 2 can be assigned to the critical items and statements. Then some fraction of this number can be assigned to the important and needed but not important items and statements. For example, 3/4 of the critical importance might be assigned to important items and statements and 1/2 of the critical importance might be assigned to needed but not important items.

[14] Sometimes the questions are not sufficiently precise. Solution of this problem generally requires the addition of different relations and new sentence templates.

4.3 Item Exclusion

In the revision process databases may become larger. Experts often are convinced that all items and statements that seem important to them must be added to a database. The consequence of these additions is a database that is either difficult or impossible for students to master in the allocated time. At this point, items and statements must be removed. There are two ways that this can be done. First, any items or statements that were not marked as at least needed can be removed. Second, entire categories that are not considered important or critical can be removed.

5 Summary

The Computer-Based Memorization System represents a method for using associative (semantic) nets, already widely used in Artificial Intelligence, for Computer-Aided Instruction. Three additions had to be made to associative networks to make them useful for education:

- Importance values had to be associated with items and statements to permit authors to emphasize features and to support files containing models of students' knowledge
- Pictures had to be associated with items
- Groups of anticipated answers had to be associated with items to allow a range of acceptable student answers

With these changes, CBMS appears to be an effective tool for the mastery of factual knowledge.

The Computer-Based Memorization System is one part of the Computer-Based Educational Software System (CBESS). Other parts of CBESS address vocabulary acquisition, equipment problem solving practice, instructional management, and script-based CAI. CBESS may be licensed.[15]

5.1 Acknowledgements

Drs. Henry Halff of Halff Associates and Douglas Wetzel of the Navy Personnel Research and Development Center evaluated the games and made many helpful suggestions. Richard B. Paulsen wrote the current versions of the games and Bradley N. Davis designed and implemented the libraries used by the CBMS programs.

The development of the CBMS programs was supported in part by the Navy Personnel Research and Development Center through Contract N00244-83-C-1759.

References

[1] John R. Anderson. *The Architecture of*

[15] Darbick Instructional Software Systems, P.O. Box 81157, Salt Lake City, Utah, 84108, licenses CBESS in both source and executable form for use under UNIX and MS-DOS operating system. UNIX is a registered trademark of AT&T Bell Laboratories and MS-DOS is a registered trademark of Microsoft, Inc.

Cognition. Harvard University Press, Cambridge, MA, 1983.

[2] R.C. Brandt. *Sequence Editor*. Computer Science Department, University of Utah, Salt Lake City, Utah 84112, April 1987.

[3] R.C. Brandt, L. Gay, B. Othmer, and H. Halff. *Computer-Based Memorization System Author and Instructor Manual*. Computer Science Department, University of Utah, Salt Lake City, Utah 84112, April 1987.

[4] R.C. Brandt and R.B. Paulsen. *Computer-Based Memorization System Student Manual*. Darbick Instructional Software Systems, P.O. Box 81157, Salt Lake City, Utah 84108, July 1987.

[5] A.M. Collins and M.R. Quillian. Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, 8:240–247, 1969.

[6] A.M. Crawford and J.D. Hollan. Development of a computer-based tactical training system. Special 82-, Navy Personnel Research and Development Center, July 1982. Draft Version.

[7] S.T. Dumas, G. Furnas, L.M. Gomez, and T.K. Landauer. The vocabulary problem in human system communication. *Communications of the ACM*, 30(11):964–971, November 1987.

[8] Nicholas V. Findler. *Associative Networks*. Academic Press, New York, New York, 1979.

[9] J. Grant and P. Marsden. The structure of memorized knowledge in students and clinicians: an explanation for diagnostic expertise. *Medical Education*, 21:92–98, 1987.

[10] Timothy McCandless. Computer-based tactical memorization system. Technical Note 81-8, Navy Personnel Research and Development Center, March 1981.

[11] T.O. Nelson. Repetition and depth of processing. *Journal of Verbal Learning and Verbal Behavior*, 16(2):151–171, 1977.

[12] D.E. Rumelhart and D.A. Norman. Active semantic networks as a model of human memory. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pages 450–457, 1973.